(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 095 669**
B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication of patent specification: **27.09.89**

(51) Int. Cl.4: **G 06 F 11/00**

(21) Application number: **83104965.5**

(22) Date of filing: **19.05.83**

(54) Automatically reconfigurable memory system and method therefor.

(30) Priority: **01.06.82 US 383640**

(43) Date of publication of application:
07.12.83 Bulletin 83/49

(45) Publication of the grant of the patent:
27.09.89 Bulletin 89/39

(84) Designated Contracting States:
DE FR GB

(56) References cited:
FR-A-2 322 427
US-A-3 812 336
US-A-3 814 922
US-A-3 906 200
US-A-4 064 558
IBM TECHNICAL DISCLOSURE BULLETIN, vol. 16, no. 4, September 1973, page 1245, New York, US; D.C. BOSSEN et al.: "Address reconfiguration for large-scale integrated memory yield enhancement"

(73) Proprietor: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: **Singh, Shanker**
Watch Hill Drive
Fishkill New York 12524 (US)
Inventor: **Singh, Vijendra Pal**
2 Barclay Street
Poughkeepsie New York 12601 (US)

(74) Representative: **Barth, Carl Otto et al**
IBM Deutschland GmbH Patentabteilung
Schönaicher Strasse 220
D-7030 Böblingen (DE)

(56) References cited:
IBM TECHNICAL DISCLOSURE BULLETIN, vol. 22, no. 10, March 1980, pages 4562-4563, New York, US; F.J. AICHELMANN Jr: "Preventing uncorrectable errors within a memory hierarchy"
1981 INTERNATIONAL TEST CONFERENCE, DIGEST OF PAPERS, Philadelphia, US, 27th-29th October 1981, pages 49-55, paper 3.4, IEEE, New York, US; R.C. EVANS: "Testing repairable RAMs and mostly good memories"

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European patent convention).

Courier Press, Leamington Spa, England.

EP 0 095 669 B1

Description

Background of the invention

The present invention relates to the automatic skewing of addresses in a memory to change memory words with uncorrectable errors into memory words with errors that can be corrected by the error correction code, or ECC, protecting the memory.

Error correction and detection schemes for encoding data are known to detect more errors than they are capable of correcting. For instance, a 64 data bit word can be provided with a single error correction and a double error detection capability by using eight check bits which are stored in the same word location in memory as the 64 data bits. A failure of any single one of the 72 cells which store the data and check bits can be corrected by error correcting circuitry. This same circuitry can also be used to detect double errors existing in the word but generally will not correct these double errors. That is, if a single bit fails the particular defective bit can be identified and, therefore, corrected. However, if two bits fail the occurrence of the failure can be detected but the failing bits generally cannot be pinpointed and, therefore, cannot be corrected.

The term "generally" has been used in connection with double error correction because some of the single error correction codes do correct specific types of double errors such as errors in adjacent bit positions. However, not all double errors will occur in a correctable pattern. Therefore, to repeat what has already been said, an error correction system generally speaking will detect a greater number of errors than it has the capability of automatically correcting.

To take advantage of this capacity of an error correction code to detect more errors than it can correct, Beausoleil U.S. Patent 3,644,902 suggests a means for changing errors that are detectable but uncorrectable into errors that are both detectable and correctable. In the Beausoleil patent, a memory unit is made up of a plurality of arrays each containing all the bits for one bit position in the memory unit. These arrays are each addressed through a decoder so that the proper bit of any word is selected from each array when the word is addressed. The Beausoleil patent suggests that, when multiple errors are to be avoided, circuitry be employed that permanently modifies the address supplied to the decoders to swap bits between words by physically swapping the arrays and thereby change words with uncorrectable errors into words with correctable errors.

In Bossen et al U.S. Patent 3,812,336, and in an article entitled "Address Reconfiguration for Large-Scale Integrated Memory Yield Inducement", appearing on page 1245 of the September 1973 issue of the IBM Technical Disclosure Bulletin, an address modification scheme was proposed to perform electronic swapping of memory bits. In this scheme the address supplied to the decoder of any particular bit array is modified by logic circuitry as a function of data stored in a shift register associated with the particular bit position of the words in the memory unit. The logic circuitry controlled by each of the registers includes an Exclusive OR gate for each of the inputs of the decoder of the particular bit position. Each of the Exclusive OR gates accepts one digit of the word address and the output of one of the stages of the linear feedback shift register and supplies its output to one of the inputs of the decoder. In the IBM Technical Disclosure Bulletin article, the decoder input address of the bad bit is placed in the shift register so that when the bad bit is requested bit location 0 is accessed instead. In the Bossen et al patent, a different Galois field number is stored in each of the shift registers starting with zero in the shift register of the first bit position and proceeding in the Galois field number sequence to the highest number needed in the shift register of the last bit position. Each time a multiple error is detected, each of the shift registers, except the shift register for the first bit position, is shifted one Galois number. This assures that the detected multiple error will be eliminated by scattering the bits making up the failing word. As a result of this scattering, each of the failing bits ends up in a different word changing the uncorrectable multiple error condition into a number of correctable single error conditions.

Test results pointing to the location of bad bits are used in Beausoleil U.S. patents 3,781,826 and 3,897,626 to divide chips into groups in accordance with the location of the failing bits. In said patent 3,897,626, these chips are mounted on memory cards with all chips having a defective cell in a given chip section being mounted on a corresponding section of a card. The address wiring is then used to skew the errors so that no memory word contains more than one bad bit. If a failure is detected by an ECC system, an Exclusive ORing of two sections of the address of the failing word will locate the bad or suspicious bit.

In European patent application 83102354.4, published as EP—A—090219 on 05.10.83, entiled "Memory System Restructed by Deterministic Permutation Algorithm", the swapping of bits between different words of a memory is accomplished by using data on bad bits in the memory. The selection of a permutation of the bit addresses is done by an exclusionary process which identifies address combinations which result in alignment of bit failures that are uncorrectable by the error correction system of the memory and then limiting the selection process to other combinations. In the preferred embodiment, failures are categorized by type, such as chip, line or bit failure to determine uncorrectable combinations of failures. The bit addresses are then permuted for successive bit position arrays in order of decreasing number of failures.

Brief description of the invention

In accordance with the present invention as claimed, the swapping of bad bits between different words of a memory is accomplished by a new process using data on bad bits in the memory. In the mentioned

European patent application EP—A—090219 the exclusionary process that selects address locations for bad bits of any bit position takes into account only the bit positions that have already had their addresses permuted. In the present arrangement, consideration is also given to data on bit failures in bit positions yet to be permuted. This is done by maintaining a list of "preferred word address locations" for the insertion of bad bits. A preferred word location is one which at that moment contains a less than a threshold level of faulty bit positions. As each faulty bit is permuted into one of these preferred word address locations, the list is changed to take into account the placement of the bad bit. Up until permutation, the unpermuted or actual physical address of a bad bit is used in calculating the list. After being permuted the logical memory address is used in changing the list.

Therefore, it is an object of the present invention to provide an improved scheme for swapping bits in memory words to change uncorrectable error conditions into correctable error conditions.

It is another object of the present invention to swap bits in memory words using fault data on bad cells in the memory.

An additional object of the invention is to swap bits in memory words based on known error conditions existing in the memory categorized by the type of error.

The drawings

These and other objects, features and advantages of the present invention can be best understood by reference to the figures of the drawings of which:

Figure 1 is a schematic of a memory employing the present invention.

Figures 2 to 6 are diagrammatic representations of a set of faults in the memory of Figure 1 prior to, during and after completion of the rearrangement process of the present invention.

Figure 7 is a flow diagram for rearranging uncorrectable errors in accordance with the present invention.

Figures 8 to 15 are diagrammatic and tabular representations of a second set of faults prior to and after the rearrangement of faults in accordance with the present invention.

Figure 16 is a block diagram of an error correcting system employing the present invention.

Detailed description of the invention

As shown in Figure 1, the storage cells 10 of each bit position B_1 to B_{72} of a plurality of 72-bit memory words are each arranged in separate identical cards 12, in a plurality of arrays or bit islands 14 on each said card. The arrays 14 are 16-bit arrays with each bit 10 located at a different intersection of one of four word lines 18 with one of four bit lines 20. The arrays 14 are each accessed through a different word decoder 22 and bit decoder 24 which receiver identical 2-bit address signals W_0, W_1 and B_0, B_1 respectively.

In addition to a word and bit decoder associated with each array each card 12 also contains a chip decoder 26 which receives a 4-bit address C'_0, C'_1, C'_2, C'_3. The chip decoder selects the output of one of the sixteen arrays 14 on each card 12.

The chip address bits C'_0, C'_1, C'_2 and C'_3 are each the output of an Exclusive OR circuit 30 that receives one address input C_0, C_1, C_2 or C_3 from the address register 32 and another input Z_0, Z_1, Z_2 or Z_3 from a different stage of a shift register 34. Therefore, if Z_0 to Z_3 are all zero, the chip decoder 26 will access the memory bit island requested by the address register 32. With any other binary combination Z_0 to Z_3 in the shift resgister 34, the chip decoder 26 will access one of the other fifteen arrays 14.

To summarize then, the memory address register transmits the same eight address bits C_0, C_1, C_2, C_3, W_0, W_1, and B_0, B_1 to all cards 12a to 12n. In each card 12, address bits W_0, W_1 and B_0, B_1 access the same cell 10 in sixteen different arrays 14. The address bits C'_0, C'_1, C'_2, C'_3, select the output of one of those chips on each card to be read out as one of the bits B_1 to B_{72} of the accessed word. If Z_0, Z_1, Z_2 and Z_3 are all zero, this will be the same bit position in the same array on all cards. If the register 34 on any card contains data other than zeros, the bit output B_i of that card will be a bit in the same position on another array 14 of the card.

In accordance with the present invention, the data placed in the registers 34 are selected by a new procedure on the basis of stored information on defects or faults of the chips. Rather than describing the process in general abstract terms, we will illustrate it with the help of an example in a step by step fashion. Consider the fault map of Figure 2 showing faults existing in the first ten bit positions of the memory of Figure 1. For purposes of the following explanation, it will be assumed that the remaining bit positions in the memory are error free. The faulty arrays or islands 14 are labelled X; their 4-bit binary addresses are denoted in hexadecimal symbols at the left hand side, under the heading W/L Address. We find that out of sixteen island words W/L, there are 2, 4, 2 and 8 words having 3, 2, 1 and 0 bit errors respectively. Let us assume that this memory is equipped with SEC/DED capability. Since there are any 8 words W/L without errors, it is possible that an excess of 8 bit errors, which are associated with 3 and 2 bit error words, can be dispersed into those errorless array words.

Step 1

Identify all the bit sections according to their dispersion capacity, that is their maximum error dispersion possibility (P_T); i.e., if we are successful in finding a suitable permutation vector, this is the maximum number of errors which can be dispersed (distributed). For example, if we can permute section 3

for bit B3 successfully, there is a potential that words 2 and 3 will have only single errors and word 9 will have been left with 2 bit errors instead of 3. Therefore, section 3 has a dispersion capacity or maximum error dispersion possibility value equal to 3. Similarly, in section 6, although there are 4 faults, with a suitable permutation vector one can disperse, at the most, 3 multiple errors. In this example, we find that sections 3, 6, 2, 5, 1, 4, 7 and 8 can potentially remove 3, 3, 2, 2, 1, 1, 1 and 1 errors respectively.

Step 2
Identify all the word addresses with zero error; i.e., word addresses 1, 4, 5, 7, 8, B, D and F.

Step 3
Also identify all word addresses corresponding to any sections which have single error/section; i.e., word addresses 3, 6 and 9; since sections 4, 7 and 8 have only a single error, located in word address 3, 6 and 9 respectively. These are defined as "Don't care word addresses" which can be gainfully used during any following iteration steps in the algorithm for finding mutually compatible permutation vectors.

Step 4
Choose a section with maximum error dispersion possibility. In our example, these are sections 3 and 6. Choose any one of these two sections. Let it be section 6. Identify faulty word addresses in section 6 i.e., word addresses 6, 9, C and E.

Step 5
EXOR each error address in section 6 with word addresses having zero error identified in step 2 and "dont care" addresses identified in step 3. Choose a mutually compatible permutation vector from the vectors obtained by EXORing. If a completely compatible vector is not found, try to also make use of some "dont care" permutation vector set.

										"Don't care"		
Error free words	⊕	1	4	5	7	8	B	D	F	3	6	9
		6	6	6	6	6	6	6	6	6	6	6
Permutation Vectors →		7	2	3	①	E	D	B	9	5	0	F
	⊕	9	9	9	9	9	9	9	9	9	9	9
Permutation Vectors →		8	D	C	E	①	2	4	6	A	F	0
	⊕	C	C	C	C	C	C	C	C	C	C	C
Permutation Vectors →		D	8	9	B	4	7	①	3	F	A	5
	⊕	E	E	E	E	E	E	E	E	E	E	E
Permutation Vectors →		F	A	B	9	6	5	3	①	D	8	7

Suppose even "don't care" sets do not help, then choose any permutation vector which removes maximum errors. For example, suppose completely compatible permutation vector 1 was not available, then we would have taken permutation vector D with help of a "don't care", or otherwise we could have chosen any of the permutation vectors among 3, 9, B because these will disperse 3 errors, but will bunch one error with a net dispersion of two errors.

Here we found vector 1 which is able to disperse all 3 multiple errors. While accomplishing this, the error free words 7, 8, D and F are used up. Since there was only a single error in address E, this single error moves to address F leaving word E error free. The memory map after permuting addresses in section 6 is shown in Figure 3 with updated P_T and errors in words.

Step 6
Now we proceed with the next section with maximum error dispersion possibility (P_T) and repeat the procedure described in step 5 with the remaining error free word addresses. The reader should note that step 5 may change the value of maximum error dispersion possibility; therefore, the next section must be chosen on the basis of updated values of P_T. We find it is section 3. Faulty words in section 3 are 2, 3 and 9.

							"Don't care"		
Error free words		1	4	5	B	E	3	6	9
	⊕	2	2	2	2	2	2	2	2
Permutation vectors		3	6	⑦	9	C	1	4	B
	⊕	3	3	3	3	3	3	3	3
Permutation vectors		2	⑦	6	8	D	0	5	A
	⊕	9	9	9	9	9	9	9	9
Permutation vectors		8	D	C	2	⑦	A	F	0

Here we find a mutually compatible permutation vector 7, which is able to disperse 3 errors. In the process of doing so, we use error free words 4, 5 and E. The updated error map with P_T is shown in Figure 4.

Step 7

Now we find that maximum error dispersion possibility P_T of each of remaining sections 1, 2, 5 and 7 is 1. In this situation, it is preferable to work with sections with a minimum number of faults. Since sections 1 and 2 have 2 faults, one can choose any of these. Let us choose section 2. Faulty words in section 2 are 0 and 2. Repeating once again as in step 6 with remaining error free words 1 and B we have:

				"Don't care"		
Error free words		1	B	3	6	9
	⊕	0	0	0	0	0
Permutation vectors		1	B	3	6	9
	⊕	2	2	2	2	2
Permutation vectors		3	9	1	4	B

Here we take advantage of "Don't care" addresses to choose permutation vector 1. Although it disperses one error (from word 0 to 1) and at the same time adds 1 error (from word 2 to 3), this new error can be dispersed by choosing section 4 with single fault which can be displaced to whatever error free word address is available. One can also choose permutation vector 3, 9 or B to provide an acceptable situation. The result of step 7 is shown in Figure 5.

Step 8

Finally, since section 7 has only one fault and it combines with the fault in section 5 with 2 faults, section 7 is the next suitable section to work with. The selection of a permutation vector is similar to that for section 4 described in step 7.

Therefore, the complete set of permutation vectors which will disperse all the errors is as follows:

Section	Permutation vectors
2	1
3	7
4	1
6	1
7	D

The reader should note that in step 7, if we had chosen section 1 instead of section 2, then permutation vector 1 would have been found to be mutually compatible without the need for don't care addresses. The permutation vector 1 for section 1 would have permuted errors in word 0 and A to 1 and B words, leaving section 4 as it was. The final memory map is shown in Figure 6, where all multi-bit errors have been

dispersed. Figure 7 is a flow diagram of the algorithm that has just been described. It shows a further modification wherein a small number of iterations is predetermined and the algorithm, if unsuccessful at first, keeps trying unless the count of UEs did not decrease at all.

The algorithm has been described in connection with a memory equipped with single error correction, double error detection (SEC/DED) capability. Figure 7 is a flow diagram for the algorithm. It can also be used to disperse errors in a memory equipped with a double error correction, triple error detection (DEC/TED) capability. For DEC/TED equipped memory, one must identify error words with three or more error addresses. The address permutation is then used to disperse those errors into words with 0 or 1 errors Permutation is done in the order determined by the highest value of updated P_T as explained earlier.

The above description of the present illustration was done without regard to the type of fault that has occurred in the arrays 14. That is, the faults were not characterized as to whether they were bit failures, line failures or array failures. In the description of the invention in connection with Figures 8 to 15 the type of fault will be categorized and will be applied to a more complex memory. The memory consists of eighteen cards, each card being populated with 128, 64K chips. These chips are placed as 32×4 array on each card. Each card contributes four bits to each 72 bit wide memory word available to the system, thereby providing 2 million, 72-bit memory words (Base Storage Module BSM). However, from error dispersion logic implementation view-point, the memory can be visualized as made up of 72 sections. Each section is equipped with its own independent address translation logic of five EX-OR gates and five latches. This logic is wired in such a fashion that the addressing bits which identify 32 chip rows can be translated according to the values of permutation bits stored earlier in the latches.

Figure 8 represents the fault map for the memory. In general, the memory consist of a 32×72 chips matrix. In the example of Figure 8, there are shown only the 52 sections with faulty arrays. The fault type in each section can be a complete chip kill, one or more bit line kill, one or more word line kill or one or more cell kill or the combination bit line, word line and cell kill type faults. Whenever the occurrence of such faults in two or more sections results in a mutual address alignment, two or more bit errors per memory word can occur. The objective of address translation is to avoid such alignments and to provide a configuration where two or more bit faults can be avoided. Entries '0', '1', '2', '3' and '4' in Figure 8 represent 'no fault', 'chip kill', 'bit line kill', 'word line kill' and 'cell kill' type faults respectively. Entries such as 5, 6, 7 and 9 represent multiple fault type situations. For example, 5 implies a chip containing a failed bit line and word line, while 7 represents a chip containing a failed word line and a cell kill. The actual addresses of a faulty bit line, word line and cell along with the information available in the fault map such as shown in Figure 8 is called a complete fault map.

The error dispersion algorithm will now be described in a step by step fashion, by using a complete fault map. The algorithm will then be applied to a partial fault map.

Let us assume the availability of a complete fault map, i.e., fault data such as shown in Figure 8 and summarized in Figure 9. Although the address of every fault type in each chip in Figure 8 is available, only addresses of faults which must be taken into account by the algorithm are tabulated in Figure 9. This is because only these faults can potentially cause alignments due to their identical "addresses" or "address components". Figure 10 lists the permutation vectors which must be excluded in case the algorithm randomly selects them from various available choices. For example, if the algorithm chooses a vector p1 for section 14, then permutation vector ($p1 \oplus 4$) should be excluded while selecting a vector for section 28. Otherwise, the bit line kill in row address 13 will align with the cell kill in row address 9 after fault dispersion and thereby will cause one two bit error word.

Figure 11 represents the Initial Error Summary Table which is basically extracted from Figure 8 and Figure 9. It consists of 32 row addresses and seven columns. The second column represents the number of faults in each row address. The 3rd, 4th, 5th and 6th columns represent the number of chip, bit line, word line and cell kill faults for each row address. Column 7 indicates the number of memory words which have two or more bit errors. The circled entries in Figure 8 represent the faults with identical address and thereby causing fault alignment. For example in row address 3, sections 20 and 28 have two chips with faulty bit lines whose addresses are identical and therefore produce 256 memory words each having two bit errors.

Step 1

With the help of Figures 8, 9 and 10 first extract the "work matrix", i.e., only those sections which cause fault alignments and thus produce two or more bit error memory words. For example, in row address 17, a chip kill in section 42 aligns with chips in sections 6, 8, 41 and 71, which have a (word line+cell) fail, a word line fail, bit line fail and a cell fail respectively. These fault alignments cause 768 memory words with two or more bit errors. Therefore, sections 6, 8, 41, 42 and 71 are included in the work matrix.

In the memory example, the work matrix is a 32×24 matrix of the following sections:

1, 3, 6, 8, 11, 14, 18, 19, 20, 25, 28, 29,
32, 36, 41, 42, 47, 48, 51, 54, 63, 65, 71.

Step 2

With the help of Figure 11, select a set of "target addresses" for each fault type, where a "target address" is an address to which a particular fault type can be moved without causing an increase in the number of multi-bit error words.

In the example, the following sets of addresses make the target address set for each fault type.

TARGET ADDRESS SET FOR CHIP KILL
1st Choice:
{Row addresses with no fault}={nil}
2nd Choice:
{Row addresses with only cell kill faults or row addresses with no faults}
{2, 7, 12, 19, 20, 21, 26, 27, 28}

TARGET ADDRESS SET FOR BIT LINE KILL
1st Choice:
{Row addresses with bit line kill or with bit line and cell kill}
{3, 4, 8, 11, 14, 15, 18, 30, 31}
2nd Choice:
{Row addresses with bit line or cell kill}
{2, 3, 4, 7, 8, 11, 12, 14, 15, 18, 19, 20, 21, 26, 27, 28, 30, 31}

TARGET ADDRESS SET FOR WORD LINE KILL
1st Choice:
{Row addresses with word line kill or with word line and cell kill}
{0, 1, 5, 10, 22, 24, 29}
2nd Choice:
{Row addresses with word line or cell kill}
{0, 1, 2, 5, 7, 10, 12, 19, 20, 21, 22, 24, 26, 27, 28, 29}

TARGET ADDRESS SET FOR CELL KILL
{2nd Choice target address set for bit line kill} or
{2nd Choice target address set for word line kill}
{0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 14, 15, 18, 19, 20, 21, 22, 24, 26, 27, 28, 29, 30, 31}

In the initial situation of Figure 8, there are no row addresses with zero faults and thus ideally there are no target addresses to which a chip kill can be moved. In such a situation the next best available target addresses can be the row addresses which have cell kill faults only. In the same way one would like to choose target addresses for bit line and word line kills such that they already have bit line and word line fails respectively. This would enable to keep the target address set size approximately the same. This will be independent of the number of faults dispersed, and therefore will help to disperse more faults.

Step 3

For each section in the "work matrix" find a maximum mutually compatible permutation vector, by EXORing each fault-type address with the addresses in their respective "target set". For example, for section 8 of the "work matrix" with cell fails in row addresses 5 and 10 and a word line fail in row address 17, the maximum mutually compatible permutation vector is determined as follows:

[Addresses in the target set for cell kill]⊕[Address of the cell kills in section 8],

[Addresses in the target set for word line kill]⊕[Address of word line kill in section 8].

0,1,2,3,4,5,7,8,10,11,12,14,15,18,19,20,21,22,24,26,27,28,29,30,31

⊕ ⊕

5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5

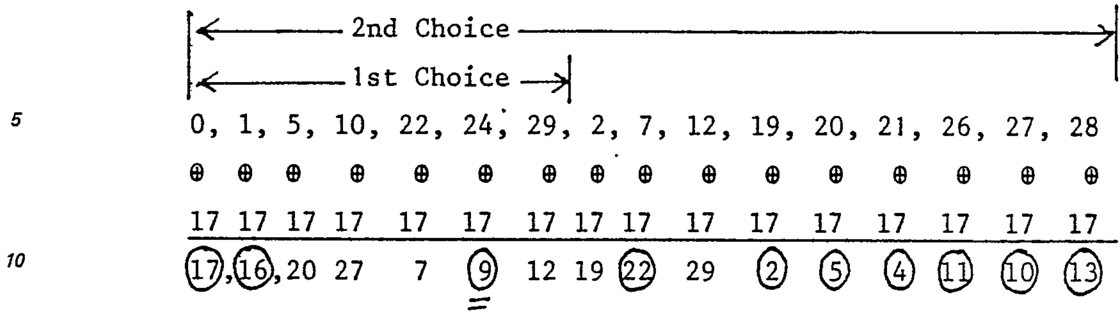
(5),(4),7,6,1,0,(2),(13),15,14,(9),(11) (10) 23 (22) (17) (16) 19 29 31 30 25 24 27 26

0,1 2 3 4 5 7 8 10 11 12 14 15 18 19 20 21 22 24 26 27 28 29 30 31

⊕ ⊕

1010101010101010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

(10)(11) 8 (9)1415(13)(2) 0 1 6 (4) (5) 24 25 30 31 28 18 (16) (17) (22) 23 20 21



Here one can easily pick a set of permutation vectors which are not only maximum mutually compatible but even completely (3-way) compatible. These are shown by circled entries. Any one of these will reduce the number of multiple bit error words. Therefore, the algorithm picks one in a random fashion. A maximum mutually compatible permutation vector is not necessarily completely compatible, however a completely compatible permutation vector is always maximum mutually compatible. For example, suppose a completely compatible vector could not be found for section 8, the algorithm will search for a two-way compatibility. In such a situation, if there is more than one choice available, the algorithm will pick that permutation vector from the choice set which decreases the number of multi-bit error words to minimum. As the algorithm continues to operate from section to section, the target address sets are continuously updated. Once a fault pair is realigned, the algorithm automatically skips the 2nd section in the pair. These characteristics of the algorithm make it highly convergent. When choosing a permutation vector the algorithm will avoid a value which is to be excluded according to Figure 10.

Figure 12 and Figure 13 show the final memory fault map and its corresponding error summary table after fault dispersion. The error column in Figure 13 indicates 0s in every row address, implying that all the faults which caused two or more bit error memory words have been dispersed.

For the sake of completeness, it is worthwhile to describe a general approach for fault dispersion with the availability of a partial fault map only. The nature of the fault map may vary with the amount of information available about the fault types and their respective addresses within individual chips. For example, a fault map may only identify chip kill, line kill and cell kill, i.e. it does not provide any information if the line killed is a word line or bit line. On the other hand, a second fault map may identify chip kill, bit line kill, word line kill and cell kill type faults, but not the location of failed bit line, word line or cell within a chip. It is obvious that it will be less tedious and less time consuming to work with the second map, however, the general approach of using these maps must be the same.

The general approach has been to start with the available fault information and disperse all possible faults, as if all the bit line, word line and cell faults have different addresses within the invididual chips. In other words, any two bit line and cells can be bunched in any of the permutable row addresses. Similarly, any two word lines and cells can be bunched in any of the permutable row addresses. This is followed by reconfiguring the memory according to the values of permutation vectors and testing for two or more bit error words. The addresses of the faulty words and the location of faulty bits within the words in the reconfigured memory provide the information about some of the fault types and their addresses not available earlier. With this new information, the fault map is updated and once again a new set of permutation vectors are found. Once again the memory is reconfigured and tested. This procedure is repeated until no multiple bit error words are found or the number of multi-bit error words cannot be further reduced.

The algorithm assumes the availability of a fault map and its corresponding error summary table of Figure 8 and Figure 11 respectively. The only difference is that the information provided in Figure 10 and hence the list of permutation vectors to be excluded from the selection is not available to the algorithm. Based on this partial information and assuming that all bit line and word line and cell kill addresses are unique, the algorithm calculates a list of permutation vectors which must disperse all the faults to a new reconfigured memory map of Figure 14 and its associated summary table of Figure 15. These permutation vectors with their respective sections are listed as follows:

Sections ⟶ 1, 6, 8, 11, 13, 14, 17, 18, 21, 25, 28, 29, 32, 36, 42, 48, 51, 56

Permutation Vectors } ⟶ 7, 20, 20, 7, 7, 9 16 10 23 17 7 29 20 3 11 27 12 27

The reconfigured memory map of Figure 14 when tested is found to contain one, two bit error word resulting from an alignment in row address 22 due to a cell kill in section 14 and a word line kill in section 48. The alignment resulted due to a common address component of a cell kill and word line kill (cf. the 3rd line of entries in the table of Figure 9 and Figure 10). Therefore, permutation vectors 9 and 27 for sections 14 and 48 are not mutually acceptable. In this example, the algorithm finds another permutation vector 21 for

section 48. This permutation vector reconfigures the memory as shown in Figure 12, which is the same as arrived at previously with a complete fault map. This memory reconfiguration does not have any two bit or multi-bit errors in any of its memory words.

Although the step by step description of the method may give the appearance that the algorithm must proceed sequentially column by column in determining permutation vectors, this is not always necessary. Many sections can be simultaneously handled if they do not have chip kill faults and the 1st choice "target address" sets for bit line, word line, and cell fails can meet the maximum comptability criterion. The other sections which either have a chip kill or need second choice "target address sets" and thus require an update on target address sets must be dealt with in a sequential operation. Such an approach can speed up algorithm execution up to 40% over a completely sequential operation.

In Figure 16 a memory 40 such as that shown in Figure 1 is checked by conventional error correction apparatus 42. The occurrence of an uncorrectable error (UE) signal from the apparatus initiates testing of the memory array by tester 44. The tester is a device for application of test patterns to the memory location containing the UE. For instance, the tester could apply a pattern of all 1's followed by a pattern of all zeros to the flawed memory location to determine bits stuck at 0 or 1 respectively. When the faulty bits have been identified, their address is stored in memory in a fault map in the manner described in the present application or an other mapping suitable for the purpose. Generation of a fault map is described in European patent application EP—A—0096030 entitled "Apparatus for High Speed Fault Mapping of Large Memories".

The UE condition also initiates operation of the permutation generation logic 48 to change data in the registers 34 of the memory to eliminate the UE condition. In accordance with the present invention, the permutation generator is a microcoded processor 48 capable of executing the algorithm set forth above.

The output of the permutation generator is the CR values for the various bit positions of the memory so these CR values are fed into the latches 34. The latches can be stages of an LSSD shift register. So the data could be shifted along the LSSD chain into the proper stages.

Claims

1. A method for automatically restructuring a memory system (40) made up of logical data words each with bit positions (12) accessed by the same logical address bits (C0—C3) through separate permuting means (30, 34) each of which converts a logical address to a separate actual address (C0'—C3') for a data bit on the basis of a separate permutation vector (Z0—Z3) that is being selected to distribute faulty data bits among the logical data words,

characterized by the following steps of:

(a) scanning a fault data map data base (Figure 2)

(a1) identifying preferred logical address locations (1, 4, 5, 7, 8, D, F; 3, 6, 9) having a number of faulty bit positions less than a number causing uncorrectable errors and

(a2) identifying all the bit positions of the individual logical address locations according to their maximum error distribution possibility (P_T);

(b) choosing one of several possible bit positions with maximum error distribution possibility and identifying all the respective logical addresses of the faulty words (6, 9, C, E);

(c) comparing the word address for the locations determined in step (a1) with the logical addresses of the faulty words determined in step (b), to thereby derive a number of potential permutation vectors;

(d) selecting for each bit position the permutation vector (1) which places the most faults in preferred word locations; and

(e) if necessary repeating the comparison and selecting steps (c) and (d) until all uncorrectable errors are permuted.

2. The memory restructuring method of claim 1 wherein said comparison is an Exclusive ORing of the word address for the locations determined in step (a1) with the logical addresses of faulty words determined in step (b) and the selection is a choice of the most common result as the permutation vector.

3. The method of claim 1 or 2 wherein the selecting of a permutation vector starts with bit positions having the most faults still conflicting with faults in other bit positions and proceeds in order of decreasing number of such conflicts.

4. The method of any preceding claim wherein the number of errors in a word is counted over the permuted bits whose logical address includes said memory word and unpermuted bits whose real address is said memory word.

5. The method of a preceding claim wherein the preferred logical address locations include those address locations (1, 4, 5, 7, 8, B, D, F) that have no faulty bits therein.

6. The method of a preceding claim wherein the preferred logical address locations include separate sets of target locations being preferred for placing different types of faults therein.

7. The method of claim 6 wherein separate sets of preferred logical address locations are identified for placing chip kill, bit line kill, word line kill and/or cell kill faults therein.

8. The method of a preceding claim wherein a set of preferred locations is split between target locations of first choice, (1, 4, 5, 7, 8, B, D, F) and target locations of second choice (3, 6, 9) or "don't care"

locations, where the second category can contribute less efficiently or only indirectly to final fault dispersion.

9. The method of a preceding claim wherein identifying preferred logical address locations includes identifying for exclusion from the preference those combinations of bit addresses that would result in combining the failures so that there are more errors in any memory word than would be correctable by the error correction code monitoring the memory.

10. A memory system (40) for carrying out the method of a preceding claim including error correction coding means (42) also detecting uncorrectable errors, testing means (44) identifying the addresses and types of the faults causing said detected uncorrectable errors, memory means (46) storing fault map data relating to said memory system (40), and permutation generation logic means (48) to select said permutation bits for dispersion of said faults on the basis of said stored fault map data, characterized by:

said permutation generation logic means comprising means comparing the actual address of each fault in each bit position with the logical address of a number of preferred word locations for placing a fault, means selecting the permutation bits which place the most faults in preferred word locations, and

means causing a repetition of the comparison and selecting steps until all uncorrectable errors are permuted.

Patentansprüche

1. Verfahren zum automatischen Restrukturieren eines Speichersystems (40) aus logischen Datenwörtern, auf deren Bitpositionen (12) von denselben logischen Adreßbits (C0—C3) durch getrennte Permutationsmittel (30, 34) Zugriff genommen wird, die jeweils eine logische Adresse in eine getrennte aktuelle Adresse (C0'—C3') für ein Datenbit auf der Basis eines getrennten Permutationsvektors (Z0—Z3) umwandeln, der zum Verteilen fehlerhafter Datenbits unter den logischen Datenwörtern aufgerufen wird,

gekennzeichnet durch die folgenden Schritte:

(a) Abtasten einer Fehlerdatenbank (Fig. 2)

(a1) Identifizieren bevorzugter logischer Adreßpositionen (1, 4, 5, 7, 8, D, F; 3, 6, 9) mit einer Anzahl fehlerhafter Bitpositionen, die niedriger ist als eine, unkorrigierbare Fehler verursachende Anzahl, und

(a2) Identifizieren aller Bitpositionen der einzelnen logischen Adreßpositionen nach deren maximal möglichen (P_T) Fehlerverteilung;

(b) Auswählen einer von mehreren möglichen Bitpositionen mit maximal möglicher Fehlerverteilung, und Identifizieren aller jeweiligen logischen Adressen der fehlerhaften Wörter (6, 9, C, E);

(c) Vergleichen der für die in Schritt (a1) bestimmten Positionen vorgesehene Wortadresse mit den logischen Adressen der in Schritt (b) bestimmten fehlerhaften Wörter, um dadurch eine Anzahl von potentiellen Permutationsvektoren abzuleiten;

(d) Auswählen des Permutationsvektors (1) für jede Bitposition, der die meisten Fehler in bevorzugte Wortpositionen stellt; und,

(e) wenn erforderlich, Wiederholen des Vergleichs und Aufrufen der Schritte (c) und (d), bis alle nicht korrigierbaren Fehler permutiert sind.

2. Speicherrestrukturierungsverfahren nach Anspruch 1, bei welchem es sich bei dem Vergleich um eine EXKLUSIV ODER-Verknüpfung der Wortadresse für die in Schritt (a1) bestimmten Positionen mit den logischen Adressen von in Schritt (b) bestimmten fehlerhaften Wörten handelt, und bei welchem das Aufrufen eine Auswahl des häufigsten Resultats als Permutationsvektor ist.

3. Verfahren nach Anspruch 1 oder 2, bei welchem die Auswahl des Permutationsvektors damit beginnt, daß sich Bitpositionen mit der größten Fehlerzahl noch mit Fehlern in anderen Bitpositionen im Konflikt befinden, und daß es nach der abnehmenden Anzahl solcher Konflikte fortgesetzt wird.

4. Verfahren nach einem der vorhergehenden Ansprüche, bei welchem die Anzahl von Fehlern in einem Wort über die permutierten Bits gezählt wird deren logische Adresse das Speicherwort enthält, und über nicht permutierte Bits, deren wirkliche Adresse das Speicherwort ist.

5. Verfahren nach einem der vorhergehenden Ansprüche, bei welchem die bevorzugten logischen Adreßpositionen diejenigen (1, 4, 5, 7, 8, B, D, F) einschließen, die keine fehlerhaften Bits enthalten.

6. Verfahren nach einem der vorhergehenden Ansprüche, bei welchem die bevorzugten logischen Adreßpositionen getrennte Gruppen von Zielpositionen enthalten, in die bevorzugt unterschiedliche Fehlerarten eingelesen werden.

7. Verfahren nach Anspruch 6, bei welchem getrennte Gruppen bevorzugter logischer Adreßpositionen zur Aufnahme von chipstörenden, bit- und wortleitungsstörenden und/oder zellstörenden Fehlern identifiziert werden.

8. Verfahren nach einem der vorhergehenden Ansprüche, bei welchem eine Gruppe von bevorzugten Positionen zwischen Zielpositionen einer ersten Auswahl (1, 4, 5, 7, 8, B, D, F) und Zielpositionen einer zweiten Auswahl (3, 6, 9) oder "don't care"-Positionen aufgespaltet wird, wobei die zweite Kategorie weniger effizient oder nur indirekt zur endgültigen Fehlerbeseitigung beitragen kann.

9. Verfahren nach einem der vorhergehenden Ansprüche, bei welchem das Identifizieren bevorzugter logischer Adreßpositionen die Indentifizierung zum Ausschluß aus der Präferenzsliste derjenigen Kombinationen von Bitadressen umfaßt, in welchem die Fehler kombiniert werden würden, so daß mehr

Fehler in einem Speicherwort sind, als durch den den Speicher überwachenden Fehlerkorrekturcode möglich wäre.

10. Speichersystem (40) zum Ausführen des Verfahrens nach einem der vorhergehenden Ansprüche, mit Fehlerkorrekturcodemitteln (42), die auch nicht korrigierbare Fehler erkennen, mit Testmitteln (44) zum
5 Identifizieren der die festgestellten nicht korrigierbaren Fehler verursachenden Adressen und Fehlerarten, mit einem Speicher (46) zum Speichern einer sich auf das Speichersystem (40) beziehenden Fehlerverteilungsliste, und mit logischen Permutationsmitteln (48) zum Aufrufen der Permutationsbits zum Beseitigen der Fehler auf der Basis der gespeicherten Fehlerverteilung,

dadurch gekennzeichnet, daß

10 die logischen Permutationsmittel enthalten: Mittel zum Vergleichen der echten Adresse jedes Fehlers in jeder Bitposition mit der logischen Adresse einer Anzahl von bevorzugten Wortpositionen zum Auffinden eines Fehlers, Mittel zum Aufrufen der Permutationsbits, welche die meisten Fehler in bevorzugten Wortpositionen auffinden, und

Mittel zum Wiederholen der Vergleichs- und Selektionsschritte, bis alle nicht korrigierbaren Fehler
15 permutiert sind.

Revendications

1. Procédé pour restructurer automatiquement un système de mémoire (40) constitué de mots de
20 données logiques ayant chacun des positions de bit (12) accédées par les mêmes bits a'adresse logique (C0—C3) par l'intermédiaire de moyens de permutation séparés (30, 34) dont chacun convertit une adresse logique en une adresse réelle séparée (C0'—C3') pour un bit de données, sur la base d'un vecteur de permutation distinct (Z0—Z3) qui est choisi pour répartir les bits de données défectueux parmi les mots de données logiques,

25 caractérisé par les opérations suivantes de:

(a) balayage d'une base de données d'implantation de données défectueuses (figure 2),

(a1) identification d'emplacements d'adresse logique préférés (1, 4, 5, 7, 8, D, F; 3, 6, 9) ayant un nombre de positions de bit défectueux inférieur au nombre engendrant des erreurs non corrigibles, et

(a2) identification de toutes les positions de bit des emplacements d'adresse logique individuels en
30 fonction de leur possibilité maximale de répartition d'erreur (P_T);

(b) choix d'une de plusieurs positions de bit possibles avec une possibilité maximale de répartition d'erreur, et identification de toutes les adresses logiques respectives des mots défectueux (6, 9, C, E);

(c) comparaison de l'adresse de mot pour les emplacements déterminés dans l'opération (a1) avec les adresses logiques des mots défectueux déterminées dans l'opération (b), de manière à obtenir un certain
35 nombre de vecteurs de permutation potentiels;

(d) sélection, pour chaque position de bit, du vecteur de permutation (1) qui place le plus grand nombre de défauts dans les emplacements de mots préférés; et

(e) si nécessaire, répétition des opérations de comparaison et de sélection (c) et (d), jusqu'à ce que toutes les erreurs non corrigibles soient permutées.

40 2. Procédé de restructuration de mémoire suivant la revendication 1, dans lequel ladite comparaison est une opération OU Exclusif de l'adresse de mot, pour les emplacements déterminés dans l'opération (a1), avec les adresses logiques des mots défectueux déterminées dans l'opération (b), et

la sélection est un choix du résultat le plus commun, comme vecteur de permutation.

3. Procédé suivant la revendication 1 ou 2, dans lequel la sélection d'un vecteur de permutation
45 commence aux positions de bit ayant le plus grand nombre de défauts encore en conflit avec des défauts dans d'autres positions de bit, et elle se poursuit dans l'ordre du nombre décroissant de ces conflits.

4. Procédé suivant l'une quelconque des revendications précédentes, dans lequel le nombre d'erreurs dans un mot est compté par rapport aux bits permutés dont l'adresse logique comprend ledit mot de mémoire et aux bits non permutés dont l'adresse réelle est ledit mot de mémoire.

50 5. Procédé suivant l'une quelconque des revendications précédentes, dans lequel les emplacements d'adresse logique préférés comprennent les emplacements d'adresse (1, 4, 5, 7, 8, B, D, F) qui ne contiennent pas de bits défectueux.

6. Procédé suivant l'une quelconque des revendications précédentes, dans lequel les emplacements d'adresse logique préférés comprennent des groupes séparés d'emplacements cibles qui sont préférés
55 pour y placer des types différents de défauts.

7. Procédé suivant la revendication 6, dans lequel des groupes séparés d'emplacements d'adresse logique préférés sont identifiés pour y placer des défauts d'élimination de puce, d'élimination de ligne de bit, d'élimination de ligne de mot et/ou d'élimination de cellule.

8. Procédé suivant l'une quelconque des revendications précédentes, dans lequel un groupe
60 d'emplacements préférés est divisé entre des emplacements cibles de premier choix (1, 5, 7, 8, B, D, F) et des emplacements cibles de deuxième choix (3, 6, 9) ou des emplacements "indifférents", dans lesquels la deuxième catégorie peut contribuer moins efficacement ou seulement indirectement à la dispersion finale des défauts.

9. Procédé suivant l'une quelconque des revendications précédentes, dans lequel l'identification des
65 emplacements d'adresse logique préférés comprend l'identification, pour exclusion de la préférence, des

combinaisons d'adresses de bit qui aboutiraient à la combinaison des défauts de sorte qu'il y aurait davantage d'erreurs dans un mot de mémoire quelconque que ce qui pourrait être corrigé par le code de correction d'erreur contrôlant la mémoire.

10. Système de mémoire (40) pour la mise en oeuvre du procédé suivant l'une quelconque des revendications précédentes, comprenant des moyens (42) de code de correction d'erreur détectant également les erreurs non corrigibles, des moyens d'essai (44) identifiant les adresses et types des défauts provoquant lesdites erreurs non corrigibles détectées, des moyens de mémoire (46) stockant des données d'implantation de défaut relatives audit système de mémoire (40), et des moyens logiques (48) de génération de permutation pour choisir les dits bits de permutation pour la dispersion desdits défauts sur la base desdites données d'implantation de défaut stockées, caractérisé en ce que:

lesdits moyens logiques de génération de permutation comprennent des moyens qui comparent l'adresse effective de chaque défaut dans chaque position de bit avec l'adresse logique d'un certain nombre d'emplacements de mot préférés pour placer un défaut.

des moyens sont prévus pour la sélection des bits de permutation qui placent le plus grand nombre de défauts dans les emplacements de mots préférés, et

des moyens sont prévus pour provoquer une répétition des opérations de comparaison et de sélection jusqu'à ce que toutes les erreurs non corrigibles soient permutées.

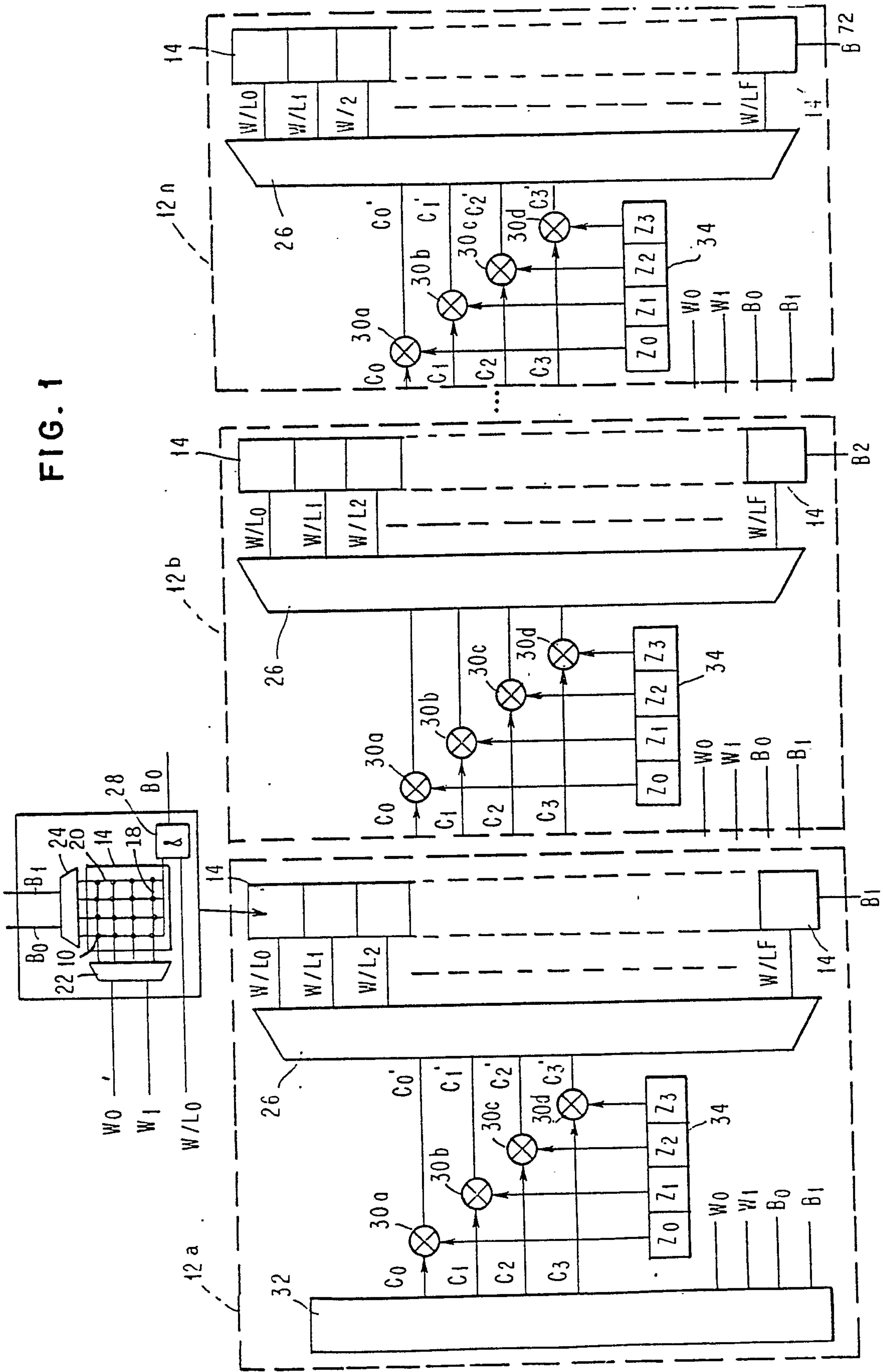


FIG. 1

FIG. 2

W/L Address \ B/L	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	# of Errors
0	X	X									2
1											
2		X	X								2
3			X	X							2
4											
5											
6					X	X	X				3
7											
8											
9			X			X		X			3
A	X										1
B											
C					X	X					2
D											
E						X					1
F											
Dispersion Capacity	1	2	3	1	2	3	1	1	0	0	

FIG. 3

W/L Address \ B/L	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Errors After Step 5
0	X	X									2
1											
2		X	X								2
3			X	X							2
4											
5											
6					X		X				2
7						X					1
8						X					1
9			X					X			2
A	X										1
B											
C					X						1
D						X					1
E											
F						X					1
After Step 5	1	2	3	1	1	0	1	1	0	0	

FIG. 4

W/L Address	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Errors After Step 6
0	X	X									2
1											
2		X									1
3				X							1
4			X								1
5			X								1
6					X		X				2
7						X					1
8						X					1
9								X			1
A	X										1
B											
C					X						1
D						X					1
E			X								1
F						X					1
After Step 6	1	1	0	0	1	0	1	0	0	0	

B/L

FI 9.81.061

FIG.5

W/L Address \ B/L	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Errors After Step 7
0	X										1
1		X									1
2											
3		X		X							2
4			X								1
5			X								1
6					X		X				2
7						X					1
8						X					1
9								X			1
A	X										1
B											
C					X						1
D						X					1
E			X								1
F						X					1
After Step 7	0	1	0	1	1	0	1	0	0	0	

FIG. 6

W/L Address \ B/L	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Errors After Step 8
0	X										1
1		X									1
2				X							1
3		X									1
4			X								1
5			X								1
6					X						1
7						X					1
8						X					1
9								X			1
A	X										1
B							X				1
C					X						1
D						X					1
E			X								1
F						X					1
After Step 8	0	0	0	0	0	0	0	0	0	0	

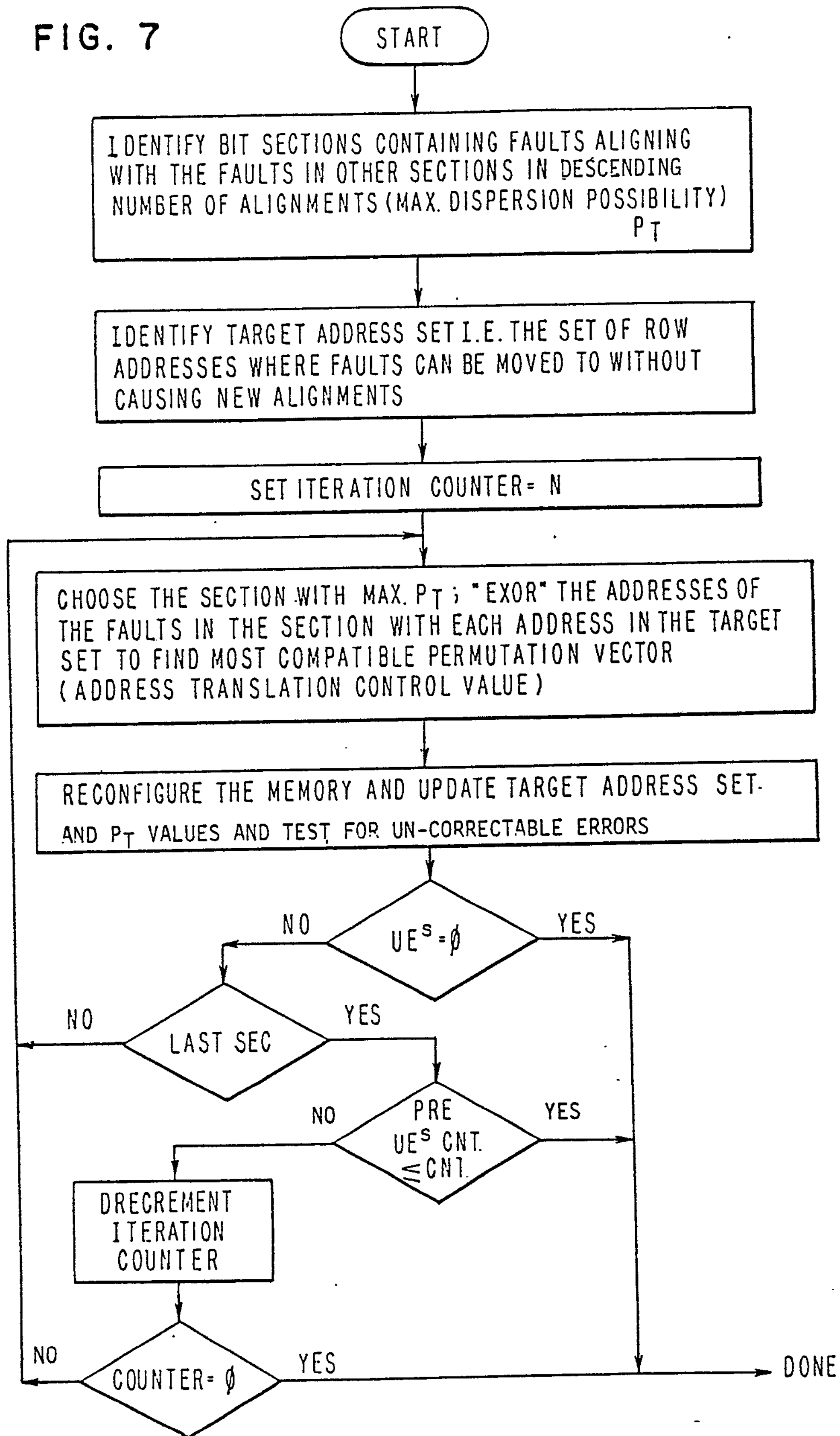


FIG. 7
START
IDENTIFY BIT SECTIONS CONTAINING FAULTS ALIGNING WITH THE FAULTS IN OTHER SECTIONS IN DESCENDING NUMBER OF ALIGNMENTS (MAX. DISPERSION POSSIBILITY) PT
IDENTIFY TARGET ADDRESS SET I.E. THE SET OF ROW ADDRESSES WHERE FAULTS CAN BE MOVED TO WITHOUT CAUSING NEW ALIGNMENTS
SET ITERATION COUNTER= N
CHOOSE THE SECTION WITH MAX. PT ; "EXOR" THE ADDRESSES OF THE FAULTS IN THE SECTION WITH EACH ADDRESS IN THE TARGET SET TO FIND MOST COMPATIBLE PERMUTATION VECTOR (ADDRESS TRANSLATION CONTROL VALUE)
RECONFIGURE THE MEMORY AND UPDATE TARGET ADDRESS SET AND PT VALUES AND TEST FOR UN-CORRECTABLE ERRORS
NO
UES=∅
YES
NO
LAST SEC
YES
NO
PRE UES CNT. ≤CNT.
YES
DRECREMENT ITERATION COUNTER
NO
COUNTER=∅
YES
DONE

FIG. 8

SECTIONS

ADDRESSES

FAULTS/SN

FIG. 9

SECTION PAIRS WITH POSSIBLE CONFLICT	PERMUTABLE ADDRESS PAIR	ADDRESS WITHIN CHIPS (WRDLINE;BITLINE) ZERO DESIGNATING "ALL"	POSSIBLE ALIGNMENT FAULT TYPES	# OF ERROR DUE TO NON-AVAILABILITY OF FAULT ADDR.
14,28	13, 9	[(0,53) ; (52,53)]	BITLINE X CELL	1
14,47	31,16	[(199,211) ; (0,211)]	CELL X BITLINE	1
14,48	31,13	[(199,211);(199,0)]	CELL X WRDLINE	1
16,20	14, 3	[(0,107) ; (0,107)]	BITLINE X BITLINE	256
20,28	3, 3	[(0,107) ; (0,107)]	BITLINE X BITLINE	256*
16,28	14, 3	[(0,107) ; (0,107)]	BITLINE X BITLINE	256
21,28	20,23	[(86,223) ; (0,223)]	CELL X BITLINE	1
29,40	13,28	[(0,213) ; (75,213)]	BITLINE X CELL	1
41,65	24,11	[(150,253) ; (0,253)]	CELL X BITLINE	1
60,65	1,11	[(245,253) ; (0,253)]	CELL X BITLINE	1
65,71	0,17	[(200,0);(200,153)]	WRDLINE X CELL	1

* # OF ERRORS IN MEMORY DUE TO 2 BITLINES HAVING SAME ADDRESS BEFORE FAULT ALIGNMENT EXCLUSION

FIG. 10

SECTION PAIRS WITH POSSIBLE CONFLICT	PERMUTABLE ADDRESS PAIR	PERMUTATION VECTOR FOR FIRST SEC IN PAIR	PERMUTATION VECTOR EXCLUDED IN SECOND SEC IN PAIR	COMMENTS
14,28	13, 9	p1	(p1 ⊕ 4)	13 ⊕ 9 = 4
14,47	31,16	p1	(p1 ⊕ 15)	31 ⊕ 16 = 15
14,48	31,13	p1	(p1 ⊕ 18)	31 ⊕ 13 = 18
16,20	14, 3	p2	(p2 ⊕ 13)	14 ⊕ 3 = 13
16,28	14, 3	p2	(p2 ⊕ 13)	14 ⊕ 3 = 13
20,28	3, 3	0	(0 ⊕ 0)	3 ⊕ 3 = 0
21,28	20,23	0	(0 ⊕ 3)	20 ⊕ 23 = 3
29,40	13,28	p3	(p3 ⊕ 17)	13 ⊕ 28 = 17
41,65	24,11	p4	(p4 ⊕ 19)	24 ⊕ 11 = 19
60,65	1,11	0	(0 ⊕ 10)	1 ⊕ 11 = 10
65,71	0,17	p5	(p5 ⊕ 17)	0 ⊕ 17 = 17

FIG. 11

ADDRESSES	FAULTS	CHIP	BIT	WORD	CELL	ERRORS
0	3	0	0	2	1	0
1	4	0	0	1	3	0
2	2	0	0	0	2	0
3	3	0	2	0	1	256
4	1	0	1	0	0	0
5	4	0	0	1	3	0
6	3	1	0	1	1	257
7	3	0	0	0	3	0
8	2	0	1	0	1	0
9	6	1	1	0	4	260
10	4	0	0	1	3	0
11	4	0	2	0	2	0
12	2	0	0	0	2	0
13	4	0	2	1	1	2
14	4	0	1	0	3	0
15	3	0	2	0	1	0
16	5	1	1	0	3	259
17	6	1	1	2	2	768
18	1	0	1	0	0	0
19	2	0	0	0	2	0
20	1	0	0	0	1	0
21	3	0	0	0	3	0
22	2	0	0	1	1	0
23	2	1	1	0	0	256
24	3	0	0	1	2	0
25	4	1	0	0	3	3
26	1	0	0	0	1	0
27	2	0	0	0	2	0
28	3	0	0	0	3	0
29	2	0	0	1	1	0
30	2	0	1	0	1	0
31	7	0	1	0	6	0
TOTAL	98	6	18	12	62	2061

FIG. 12

FIG. 13

ADDRESSES	FAULTS	CHIP	BIT	WORD	CELL	ERRORS
0	4	0	0	2	2	0
1	4	0	0	1	3	0
2	2	0	0	0	2	0
3	5	0	1	0	4	0
4	3	0	2	0	1	0
5	6	0	0	3	3	0
6	1	0	0	0	1	0
7	3	0	1	0	2	0
8	3	0	2	0	1	0
9	1	1	0	0	0	0
10	4	0	0	2	2	0
11	4	0	2	0	2	0
12	2	0	0	0	2	0
13	1	1	0	0	0	0
14	7	0	2	0	5	0
15	0	0	0	0	0	0
16	5	0	3	0	2	0
17	3	0	1	0	2	0
18	1	1	0	0	0	0
19	2	0	0	0	2	0
20	1	1	0	0	0	0
21	1	0	0	0	1	0
22	5	0	0	1	4	0
23	1	1	0	0	0	0
24	7	0	0	2	5	0
25	3	0	0	0	3	0
26	1	1	0	0	0	0
27	2	0	1	0	1	0
28	3	0	0	0	3	0
29	4	0	0	1	3	0
30	4	0	2	0	2	0
31	5	0	1	0	4	0
TOTAL	98	6	18	12	62	0

FIG. 14

SECTIONS

1 3 6 8 9 11 12 13 14 16 17 18 19 20 21 22 24 25 26 28 29 30 32 33 36 40 41 42 43 44 47 48 49 50 51 52 54 55 56 58 59 60 62 63 64 65 66 68 69 70 71 72

ADDRESSES

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

FAULTS/SN

FIG.15

ADDRESSES	FAULTS	CHIP	BIT	WORD	CELL	ERRORS
0	4	0	0	2	2	0
1	4	0	0	1	3	0
2	2	0	0	0	2	0
3	5	0	1	0	4	0
4	3	0	2	0	1	0
5	6	0	0	3	3	0
6	1	0	0	0	1	0
7	3	0	1	0	2	0
8	3	0	2	0	1	0
9	1	1	0	0	0	0
10	4	0	0	2	2	0
11	4	0	2	0	2	0
12	2	0	0	0	2	0
13	1	1	0	0	0	0
14	7	0	2	0	5	0
15	0	0	0	0	0	0
16	5	0	3	0	2	0
17	4	0	1	0	3	0
18	1	1	0	0	0	0
19	2	0	0	0	2	0
20	1	1	0	0	0	0
21	1	0	0	0	1	0
22	6	0	0	2	4	0
23	1	1	0	0	0	0
24	5	0	0	1	5	0
25	3	0	0	0	3	0
26	1	1	0	0	0	0
27	2	0	1	0	1	0
28	3	0	0	0	3	0
29	4	0	0	1	3	0
30	4	0	2	0	2	0
31	4	0	1	0	3	0
TOTAL	98	6	18	12	62	0

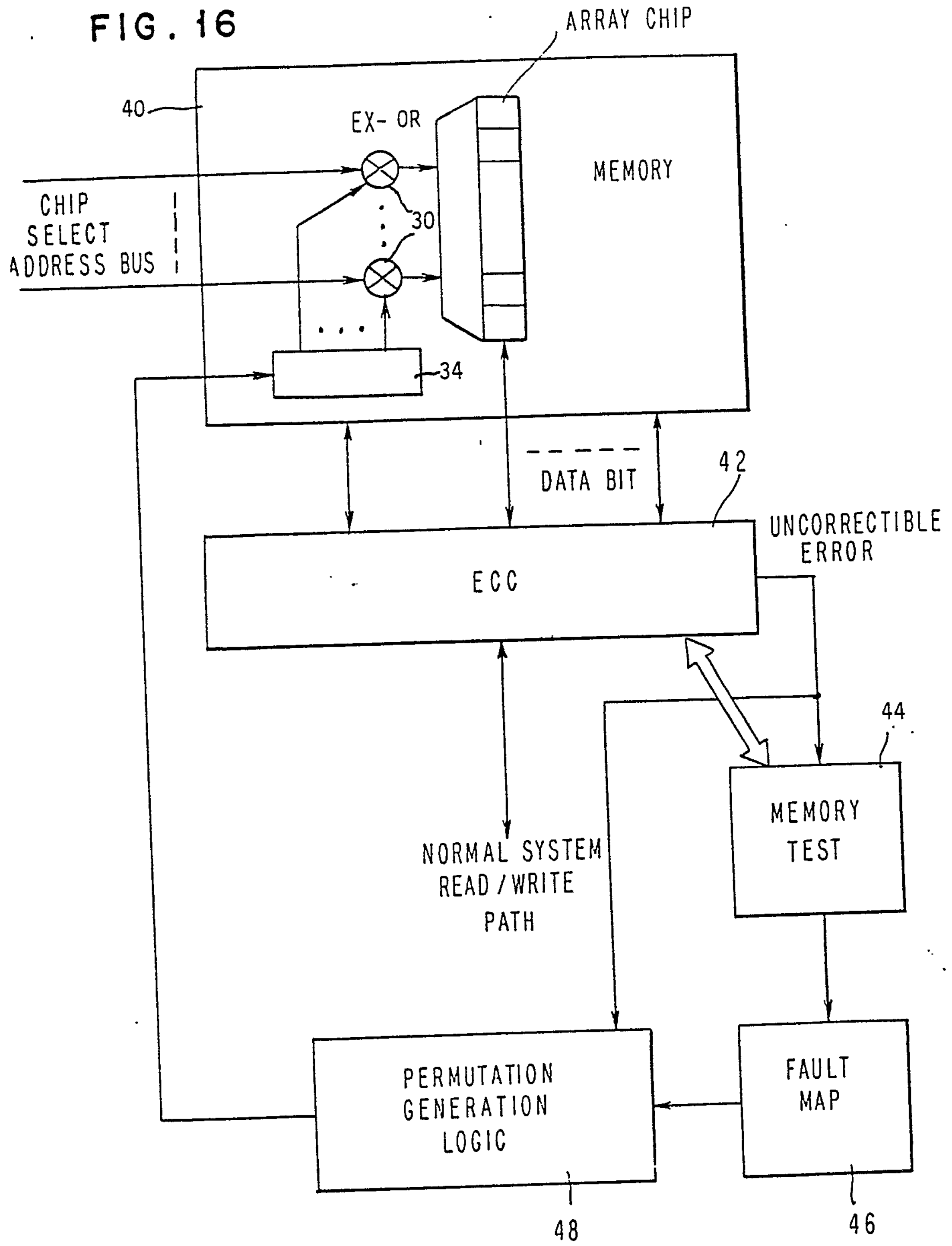


FIG. 16
ARRAY CHIP
40
EX- OR
MEMORY
CHIP
SELECT
ADDRESS BUS
30
34
DATA BIT
42
UNCORRECTIBLE
ERROR
ECC
44
MEMORY
TEST
NORMAL SYSTEM
READ / WRITE
PATH
PERMUTATION
GENERATION
LOGIC
FAULT
MAP
48
46